Amendments to the Specification

Please replace the paragraph that begins on Page 4, line 13 and carries over to Page 5, line 2 with the following marked-up replacement paragraph:

-- In some applications, it is important to speed up computation of the PRBG function. One possible way to do this is to restrict its input to small values of x. Let B be an integer bound, and assume that x must be less than B for the PRBG function y = G^x mod P. It appears to be reasonable to assume that computing the discrete logarithm of y is still hard, even knowing that x is less than some value B. Indeed, it has been demonstrated that the running time of the index calculus method depends only on the size N of the whole group ZP*. Depending on the size of B, different methods for computing the inverse of function f may actually be more efficient [[that]] <u>than</u> the index calculus method. For example, the so-called "baby-step, giant-step" algorithm or the "rho" or "lambda" algorithms can compute the discrete logarithm of y in time proportional to the square root of B. But if B is chosen large enough (e.g. if B is approximately log^2 N bits long), then this is still too much time to be considered computationally feasible and thus provides a secure PRBG. --

Please replace the paragraph that begins on Page 18, line 12 and carries over to Page 19, line 7 with the following marked-up replacement paragraph:

-- It can be demonstrated that the computation of G^x mod P with a short exponent, as disclosed herein, requires roughly (1.5 log [[x]] <u>2^x</u>) modular multiplications; when x is limited to 160 bits, as discussed above, this is equivalent to (1.5 * 160) or 240 modular multiplications. Contrast this to the cost of the Patel-Sundaram generator, where the same number of pseudo-random bits would cost (1.5 * N) or 1536 modular multiplications. The modular multiplications are the most expensive operation in the PRBG of the present invention. Because the modular exponentiations are computed over the same (fixed) basis G^, the powers of G^ can be precomputed and stored in a table to enable more quickly computing some particular G^x. If a table T stores values for G^(2^i) mod P, where i takes on the integer values from 0 through C, then on average (.5 * C) multiplications are required for computing G^ for a random C-bit exponent. The table requires on the order of (C * N) bits of memory. By using a precomputation table of this type, the PRBG of the preferred embodiment requires a table of size (160 * 1024) bits or 20 kilobytes, and the function G^x can be computed with only (.5 * 160) = 80 multiplications. (As previously discussed, the precomputation tables of Peralta and of Patel-Sundaram are 1 megabit tables. Thus, the present invention provides advantages in terms of reduced storage requirements as contrasted to those prior art PRBGs.) --

Please replace the paragraph on Page 20, lines 3 - 17 with the following marked-up replacement paragraph:

-- Refer to Figs. 3 and 4 for an illustration of operation of a preferred embodiment of the PRBG of the present invention. As has been stated, the PRBG input of the present invention is significantly shorter than the number of output bits generated per iteration. Fig 3 illustrates a single iteration of a preferred embodiment of the PRBG, using a 160-bit input value x, which is processed by the PRBG algorithm f(x) = G^x mod P to yield 1024 bits. From these 1024 generated bits, 160 are selected as input to the next iteration and [[860]] <u>864</u> are used as pseudo-random output bits. Note that the 160 selected bits are not required to be the top-most bits, nor are they required to be contiguous: various selection techniques may be used without deviating from the scope of the present invention. Fig. 4 illustrates use of multiple iterations of the PRBG to generate a sequence of pseudo-random output bits. At each iteration, the [[860]] <u>864</u> output bits are used in forming the output sequence, preferably by concatenating the groups of bits to the output sequence of a prior iteration. The iteration may be repeated as necessary, depending on the requirements of an application for which the PRBG is operating. (Typically, this PRBG will be used with encryption applications, for example to generate keying material, although this is for purposes of illustration and not of limitation.) --